数理解析研究所講究録
第1666巻 2009年 145-152

近似GCDによる人間らしい簡単化
– 整数係数の近似GCDの応用 –

長坂耕作
Kosaku Nagasaka
神戸大学人間発達環境学研究科
Graduate School of Human Development and Environment, Kobe University*

1 はじめに

本稿では，著者が2008年にISSAC2008のポスターセッション[1]で発表した整数係数多項式の近似GCDの応用として，厳密な代数処理では簡単化出来ない数式の人間らしい簡単化について取り上げる。ここで言う人間らしい簡単化は，多少の残余項を考慮した数式の簡単化，即ち，近似代数演算による数式の簡単化を意味するものとする。まず，整数係数多項式の近似GCDの定義を振り返っておく。

定義 1 (Approx. GCD Over Integers)
Let $f(\vec{x})$ and $g(\vec{x})$ be polynomials in variables $\vec{x} = x_1, \ldots, x_\ell$ over $\mathbb{Z}$, and let ε be a small positive integer. If $f(\vec{x})$ and $g(\vec{x})$ satisfy

$$f(\vec{x}) = t(\vec{x})h(\vec{x}) + \Delta_f(\vec{x}),\ \ g(\vec{x}) = s(\vec{x})h(\vec{x}) + \Delta_g(\vec{x}),\ \ \varepsilon = \max\{\|\Delta_f\|, \|\Delta_g\|\},$$

for some polynomials $\Delta_f, \Delta_g \in \mathbb{Z}[\vec{x}]$, then we say that the above polynomial $h(\vec{x})$ is an ***approximate GCD over integers****. We also say that $t(\vec{x})$ and $s(\vec{x})$ are* ***approximate cofactors over integers****, and we say that their* ***tolerance*** *is ε. ($\|p\|$ denotes a suitable norm of polynomial $p(\vec{x})$.)* ◁

例 1 (整数係数多項式の近似 GCD の例：二変数多項式の場合)
次の互いに素な整数係数多項式 $f(x_1, x_2)$ と $g(x_1, x_2)$ について，その近似 *GCD* を例示する。

$$\begin{aligned} f(x_1,x_2) &= 89x_1^2x_2^2 - 87x_1x_2^2 - 136x_2^2 + 15x_1^2x_2 + 132x_1x_2 + 119x_2 - 42x_1^2 + 166x_1 + 139, \\ g(x_1,x_2) &= 56x_1^2x_2^2 - 45x_1x_2^2 - 98x_2^2 - 13x_1^2x_2 + 46x_1x_2 + 225x_2 - 12x_1^2 + 80x_1 - 112. \end{aligned}$$

この多項式ペアは，次のように整数上の近似 *GCD* $(5x_1x_2 - 9x_2 - 3x_1 + 14)$ を持つ。下線部の数字は，整数上で *GCD* を持つために変動させた係数部を表している。整数としては僅かな違いであるが，一般の実数や複素数上の近似 *GCD* と比べ，その変動は非常に大きいことに注意されたい。

$$\begin{aligned} f(x_1,x_2) &\approx (18x_1x_2 + 15x_2 + 14x_1 + 10)(5x_1x_2 - 9x_2 - 3x_1 + 14) \\ &= \underline{90}x_1^2x_2^2 - 87x_1x_2^2 - 13\underline{5}x_2^2 + 1\underline{6}x_1^2x_2 + 13\underline{1}x_1x_2 + 1\underline{20}x_2 - 42x_1^2 + 166x_1 + 1\underline{40}, \\ g(x_1,x_2) &\approx (11x_1x_2 + 11x_2 + 4x_1 - 8)(5x_1x_2 - 9x_2 - 3x_1 + 14) \\ &= 5\underline{5}x_1^2x_2^2 - 4\underline{4}x_1x_2^2 - 9\underline{9}x_2^2 - 13x_1^2x_2 + 4\underline{5}x_1x_2 + 22\underline{6}x_2 - 12x_1^2 + 80x_1 - 112. \end{aligned}$$

◁

*nagasaka@main.h.kobe-u.ac.jp

整数係数多項式の近似 GCD は，近似代数演算を様々な代数上で可能とする取組の一環であるが，その具体的な応用例に乏しかった。本稿で取り扱う応用は，ISSAC 2008 のポスター発表での Adam Strzebonski とのやりとりの中で彼から指摘されたもので，整数上の近似 GCD や因数分解の新たな側面として，人間らしい数式の簡単化を実現しようとするものである。

2 整数係数多項式の近似 GCD アルゴリズムの復習

整数係数多項式の近似 GCD の応用として数式の簡単化を行うためには，複数多項式の近似 GCD を計算する必要があるため，多項式のペアの近似 GCD の計算について復習しておく。

2.1 多変数の部分終結式写像と GCD の関係

$Syl_r(f,g)$ を $f(\vec{x})$ と $g(\vec{x})$ に関する r 次の部分終結式写像と呼ぶ。なお，$r = 0, \ldots, \min\{n,m\}-1$ である。

$$Syl_r(f,g): \begin{array}{ccc} \mathcal{P}_{m-r-1} \times \mathcal{P}_{n-r-1} & \to & \mathcal{P}_{n+m-r-1} \\ (s(\vec{x}), t(\vec{x})) & \mapsto & s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}). \end{array}$$

ここで，$\mathcal{P}_d$ は全次数が d の多項式全体の集合とする。ただし，$f, g \in \mathbb{Z}[x_1, \ldots, x_\ell]$，$n = \mathrm{tdeg}(f)$，$m = \mathrm{tdeg}(g)$ とする。良く知られている事実部分終結式写像が単射でない最大の r に対して，$f(\vec{x})/t(\vec{x})$ と $g(\vec{x})/s(\vec{x})$ は，$f(\vec{x})$ と $g(\vec{x})$ の GCD となる。以下では，多項式 $p(\vec{x})$ の辞書式順序による係数ベクトルを $\vec{p}$ と表記する。

定義 2 (係数ベクトルの長さ)
次式で定義される $\beta_{d,r}$ を導入することで，全次数が d 以下の単項式の個数は，$\beta_{d,0}$ と表せる。

$$\beta_{d,r} = \begin{pmatrix} d-r+\ell \\ \ell \end{pmatrix}.$$

◁

定義 3 (畳み込み行列)
k 次の畳込行列 $C_k(f)$ を，全次数が $k-1$ の多項式 $p(\vec{x})$ に対して，$C_k(f)\vec{p} = \vec{fp}$ を満たすものと定義する。

◁

定義 4 (部分終結式写像の行列表現)
畳み込み行列により $f(\vec{x})$ と $g(\vec{x})$ の部分終結式写像は，次のシルベスター行列で表現される。

$$Syl_r(f,g) = (C_{m-r}(f) \mid C_{n-r}(g)) \in \mathbb{Z}^{(\beta_{n+m-1,r}) \times (\beta_{m-1,r}+\beta_{n-1,r})}.$$

◁

本来であれば，シルベスター行列から簡単に GCD の係数ベクトルを取り出せますが，近似 GCD の計算を視野に入れ，格子算法 [2] により GCD の係数ベクトルを求めることを考える。そのため，新たに次の行列 $Syl_r^E(f,g)$ を定義する。ここで，E_i は i 次の単位行列で，$c_B \in \mathbb{Z}$ とする。

$$Syl_r^E(f,g) = (E_{\beta_{n-1,r}+\beta_{m-1,r}} \mid c_B \cdot Syl_r^t(f,g)).$$

補題 5 ($Syl_r^E(f,g)$ に関する補題)
B を $f(\vec{x})$ と $g(\vec{x})$ それぞれの因子係数の上限，$c_B = 2^{(\beta_{n-1,r}+\beta_{m-1,r}-1)/2}\sqrt{\beta_{n-1,r}+\beta_{m-1,r}}B$ とする。このとき，部分終結式写像が単射でない最大の r に対して，LLL アルゴリズムは $Syl_r^E(f,g)$ の行ベクトルで生成される格子から，$f(\vec{x})$ と $g(\vec{x})$ の GCD による余因子の整数倍の係数ベクトルが，最初の $\beta_{n-1,r}+\beta_{m-1,r}$ 個の要素である短いベクトルを検出できる。 ◁

例えば，Gelfond の上限を使うと，$B = \max\{2^{n\ell}\,\|f\|_2, 2^{m\ell}\,\|g\|_2\}$ である。もっとも，格子算法は理論上の上限に比して，十分小さなベクトルをほとんどの場合計算できるため，このような大きな上限を利用しなければ GCD を計算できないことは少ない。

例 2 (格子算法による GCD の計算例)
次の多項式ペアの GCD を格子算法で計算することを考える。

$$f(x) = 49x^2 - 24 = (7x-5)(7x+5),\quad g(x) = 49x^2 + 70x + 25 = (7x+5)(7x+5).$$

$Gelfond$ 等の上限に比すとかなり小さい上限となる $c_B = 1$ に対して，$Syl_0^E(f,g)$ を構築し格子算法で短いベクトルを計算すると，次の下線部のベクトルが得られる。これら多項式の GCD の余因子の係数ベクトルが検出できていることが確認できる。

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -25 & 0 & 49 & 0 \\ 0 & 1 & 0 & 0 & 0 & -25 & 0 & 49 \\ 0 & 0 & 1 & 0 & 25 & 70 & 49 & 0 \\ 0 & 0 & 0 & 1 & 0 & 25 & 70 & 49 \end{pmatrix} \rightarrow \left(\begin{array}{cccccccc} -5 & -7 & -5 & 7 & 0 & 0 & 0 & 0 \\ \hline -2 & -3 & -2 & 3 & 0 & 10 & 14 & 0 \\ 0 & -1 & -1 & 1 & -25 & -20 & 21 & 0 \\ -2 & -2 & -2 & 3 & 0 & -15 & 14 & 49 \end{array}\right).$$

◁

2.2 整数上の近似 GCD をどう計算するか

前述の方法で余因子の係数ベクトルを求めることは出来るが，例えば次のような近似 GCD を持つ多項式ペアの場合，余因子から直接的に除算により GCD を求めることは困難である。

$$f(x) = 49x^2 - 24 = (7x-5)(7x+5) - 1,\quad g(x) = 49x^2 + 70x + 25 = (7x+5)(7x+5).$$

即ち，次のような割り切れない除算を近似 GCD の関係をなるべく保ちつつ計算する必要が生じてしまう。

$$f(x)/(7x-5) = (49x^2-24)/(7x-5),\quad g(x)/(7x+5) = (49x^2+70x+25)/(7x+5).$$

そこで，この除算も整数演算で済ますために，次の行列 $H(f,g,t,s)$ の各行ベクトルで張られる整数格子に対して，再度格子算法を適用して近似 GCD を求めることとする。ここで，s,t は近似余因子，E_i は i 次の単位行列で，$c_H \in \mathbb{Z}$ とする。

$$H(f,g,t,s) = \left(\begin{array}{c|cc} E_{\beta_{r+1,0}+1} & c_H\cdot\vec{f} & c_H\cdot\vec{g} \\ & c_H\cdot C_{r+2}(-t)^t & c_H\cdot C_{r+2}(s)^t \end{array}\right).$$

補題 6 ($H(f,g,t,s)$ に関する補題)
B を $f(\vec{x})$ と $g(\vec{x})$ それぞれの因子係数の上限，$c_H = 2^{\beta_{r+1,0}/2}\sqrt{\beta_{r+1,0}+1}B+1$ とする。このとき，部分終結式写像が単射でない最大の r に対して，格子算法は $H(f,g,t,s)$ の行ベクトルで生成される格子から，$f(\vec{x})$ と $g(\vec{x})$ の GCD の整数倍の係数ベクトルが，$2,\ldots,(\beta_{r+1,0}+1)$ 番目の要素である短いベクトルを検出できる。

例 3 (格子算法による近似 GCD の計算例)
実際に，前述の次の多項式ペアに対して近似 GCD を計算してみる。

$$f(x) = 49x^2 - 24 = (7x-5)(7x+5) - 1,\ \ g(x) = 49x^2 + 70x + 25 = (7x+5)(7x+5).$$

$Gelfond$ 等の上限に比すとかなり小さい上限となる $c_B = 1$ に対して，$Syl_0^E(f,g)$ を構築し格子算法で短いベクトルを計算すると，次の下線部のベクトルが得られる。下線部は，右側半分の剰余項に対応するサイズが最も小さいベクトルであり，この例では近似余因子の係数ベクトルに対応している。

$$\left(\begin{array}{cccc|cccc} 1&0&0&0&-24&0&49&0\\ 0&1&0&0&0&-24&0&49\\ 0&0&1&0&25&70&49&0\\ 0&0&0&1&0&25&70&49 \end{array}\right) \rightarrow \left(\begin{array}{cccc|cccc} -2&-3&-2&3&-2&7&14&0\\ -5&-7&-5&7&-5&-7&0&0\\ \hline 7&9&6&-9&-18&-21&7&0\\ 3&5&3&-4&3&-10&14&49 \end{array}\right).$$

近似余因子候補の係数ベクトルから，新たに行列 $H(f,g,t,s)$ を構築し，その行ベクトルの張る整数格子に対して格子算法を用いて，短いベクトルを計算する。ここでは，$c_H = 1$ としている。

$$\left(\begin{array}{ccc|cccccc} 1&0&0&-24&0&49&25&70&49\\ 0&1&0&5&-7&0&-5&-7&0\\ 0&0&1&0&5&-7&0&-5&-7 \end{array}\right) \rightarrow \left(\begin{array}{ccc|cccccc} 1&5&7&1&0&0&0&0&0\\ \hline 0&1&0&5&-7&0&-5&-7&0\\ 0&0&1&0&5&-7&0&-5&-7 \end{array}\right).$$

結果から近似 GCD の係数ベクトルが求まり，近似 GCD は「$7x+5$」で誤差が「1」であることがわかる。◁

2.3 検索対象を減らす方策と実際のアルゴリズム

前述の方法では，格子算法で求めた短いベクトルのうち，どのベクトルが近似余因子に該当するか実際に近似 GCD 候補を計算するまでわからない。そのため，無駄に格子算法を繰り返さなければならない。そこで，実際のアルゴリズムにおいては次のような基準を用いて候補ベクトルの数を減じている。

Criterion 1 (定数項のあるなし)
$t(\vec{x})$ と $s(\vec{x})$ を，$Syl_r^E(f,g)$ の行ベクトルから生成される格子に含まれる任意のベクトル $\vec{u}$ に対応する余因子候補とする。このとき，次式が満たされるならば，$\vec{u}$ は検索対象から除外可能。

$$t(\vec{0}) = 0 \ \textit{while}\ |f(\vec{0})| > \varepsilon \ \ \textit{or}\ \ s(\vec{0}) = 0 \ \textit{while}\ |g(\vec{0})| > \varepsilon.$$

◁

Criterion 2 (余因子の次数減少)
$t(\vec{x})$ と $s(\vec{x})$ を，$Syl_r^E(f,g)$ の行ベクトルから生成される格子に含まれる任意のベクトル $\vec{u}$ に対応する余因子候補とする。このとき，次式が満たされるならば，$\vec{u}$ は検索対象から除外可能。

$$\begin{array}{rl} & n - \mathrm{tdeg}(t) > m - \mathrm{tdeg}(s) \ \textit{while}\ \ \|\mathrm{term}_n(f)\| > \varepsilon, \\ \textit{or} & n - \mathrm{tdeg}(t) < m - \mathrm{tdeg}(s) \ \textit{while}\ \ \|\mathrm{term}_m(g)\| > \varepsilon. \end{array}$$

ここで，$\mathrm{term}_d(p)$ は全次数 d の項の総和とする。 ◁

Criterion 3 (変数毎の確認)
$t(\vec{x})$ と $s(\vec{x})$ を全次数が $n-k$ と $m-k$ である余因子候補とする。このとき，次式が満たされるならば，$\vec{u}$ は検索対象から除外可能。ここで，$\mathrm{cf}_{x_i^d}(p)$ は，多項式 $p(\vec{x})$ の項 x_i^d の係数とする。

(1) $|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)| \neq 0$ であり，

$$\exists i, \left| \frac{|\mathrm{cf}_{x_i^{n-k}}(t)\mathrm{cf}_{x_i^m}(g)| - |\mathrm{cf}_{x_i^{m-k}}(s)\mathrm{cf}_{x_i^n}(f)|}{|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)|} \right| > \varepsilon$$

$$or \quad \exists i, \left| |\mathrm{cf}_{x_i^m}(g)| - \frac{|\mathrm{cf}_{x_i^{m-k}}(s)|(|\mathrm{cf}_{x_i^n}(f)| + |\mathrm{cf}_{x_i^m}(g)|)}{|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)|} \right| > \varepsilon.$$

(2) $|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)| = 0$ かつ $\exists i, \max\{|\mathrm{cf}_{x_i^n}(f)|, |\mathrm{cf}_{x_i^m}(g)|\} > \varepsilon$. ◁

結果として，実際のアルゴリズムは次のようになる。

アルゴリズム 1 (整数係数多項式の近似 GCD -改良版 (抄)-)
Input: $f(\vec{x}), g(\vec{x}) \in \mathbb{Z}[\vec{x}]$, *of total degrees* n *and* m, *respectively.*
Output: $s(\vec{x}), t(\vec{x}), h(\vec{x}) \in \mathbb{Z}[\vec{x}]$ *satisfying* $f(\vec{x}) \approx s(\vec{x})h(\vec{x})$ *and* $g(\vec{x}) \approx t(\vec{x})h(\vec{x})$.

1. $\varepsilon \leftarrow 1$ *and while* $\varepsilon < \min\{\|f\|, \|g\|\}$ *do* **2–11**
2. $r \leftarrow \min\{n, m\} - 1$ *and while* $r \geq 0$ *do* **3–10**
3. *construct a matrix* $Syl_r^E(f, g)$ *with* $c_B = 1$
4. $c \leftarrow \max\{\|f\|, \|g\|\}$ *and while* $c \leq c_B$ *do* **5–9**
5. *multiply the right* $\beta_{n+m-1,r}$ *column vectors of* $Syl_r^E(f, g)$ *by* $\max\{\|f\|, \|g\|\}$
6. *apply the LLL algorithm to row vectors of* $Syl_r^E(f, g)$
7. *for each lattice basis vector unsatisfying the criteria, sorted by the norm of right* $\beta_{n+m-1,r}$ *columns, do* **8**
8. *apply the LLL algorithm to a matrix* $H(f, g, t, s)$, *let* $h(\vec{x}), t(\vec{x}), s(\vec{x})$ *be candidate approximate GCD and cofactors, and output* $h(\vec{x}), t(\vec{x}), s(\vec{x})$ *if* $\|f - th\| + \|g - sh\| \leq 2\varepsilon$
9. $c \leftarrow c \times \max\{\|f\|, \|g\|\}$
10. $r \leftarrow r - 1$
11. $\varepsilon \leftarrow \varepsilon \times 10$
12. *output "not found".* ◁

3 数式の簡単化への利用に向けて

まず，本稿で対象としている数式の簡単化は，次のような既約な多項式を如何に簡単化するかである。

$$\left(-2yx^2 + 2x^2 + y^2x - 2yx + 3x - y + 1\right) z^2 + \left(-4yx^2 + 4x^2 + 2y^2x - 4yx + y - 1\right) z + y - 2x.$$

既約な多項式の簡単化は難しく，例えば，数式の簡単化に関して進んでいるといわれる *Mathematica* の組込関数 FullSimplify を用いても次のようにしか変形できない。

$$-2(y-1)z(z+2)x^2 + (z(3z + (y-2)y(z+2)) - 2)x + y - z + z(-zy + y + z).$$

このような多項式に対して近似演算を導入し，次のような簡単化を行おうとするのが本稿の目的である。

$$(2x - y + 1)\left((-yx + x + 1)z^2 + (-2yx + 2x - 1)z - 1\right) + 1.$$

3.1 簡単化に必要な近似演算

前述の簡単化を行う最も率直な方法は，次式のような近似因数分解を整数係数に限定して求めることであるが，少なくとも現時点でこれを実現するアルゴリズムは知られていない。

$$f(\vec{x}) \Rightarrow g(\vec{x})h(\vec{x}) + \Delta_f(\vec{x}).$$

そこで，特定の変数に関する係数多項式のみを取り出し，その整数係数多項式としての近似 GCD を求めることで簡単化を実現する。前述の簡単化の例はこのタイプであり，次のような変形を行うことになる。

$$\sum \vec{f_i}(\vec{x})z \Rightarrow g(\vec{x})(\sum h_i(\vec{x})z) + \Delta_f(\vec{x}, z).$$

これを素直にアルゴリズムとしたものが以下である。

アルゴリズム 2 (整数係数多項式の近似 GCD による簡単化)
Input: $f(\vec{x}) \in \mathbb{Z}[\vec{x}]$.
Output: $f(\vec{x})$ を簡単化した結果.

1. $i = 1$ とする。
2. $f(\vec{x})$ を $\sum_j f_j(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell)x_i^j$ と表現する。
3. $f_j(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell)$ の近似 *GCD* を計算する。
4. 近似 *GCD*（$g(\vec{x})$ とおく）が求まれば，$g(\vec{x})(\sum h_i(\vec{x})x_i) + \Delta_f(\vec{x})$. を出力する。
5. $i = i + 1$ として，$i \leq \ell$ であれば手順 *2* へ。そうでなければ，$f(\vec{x})$ をそのまま出力する。 ◁

このアルゴリズムにより数式の簡単化を行うためには，前述の多項式ペアの近似 GCD を複数多項式の近似 GCD に拡張する必要がある。これは，自然な拡張であり可能なはずだが，これまでは試していなかった。複数多項式（$f_0(\vec{x}), \ldots, f_k(\vec{x})$）の場合は，次の部分終結式写像 $\mathcal{S}_r$ を使うことで，多項式ペアの場合と同じ方法で近似 GCD を計算することが可能である。なお，複数多項式の部分終結式による GCD の計算については，一変数多項式となるが Rupprecht[3] を参照されたい。

$$\mathcal{S}_r : \left\{ \begin{array}{ccc} \prod_{i=0}^{k} \mathcal{P}_{n_i - r - 1} & \rightarrow & \prod_{i=1}^{k} \mathcal{P}_{n_0 - n_i - r - 1}, \\ \begin{pmatrix} u_0 \\ \vdots \\ u_k \end{pmatrix} & \mapsto & \begin{pmatrix} u_1 f_0 + u_0 f_1 \\ \vdots \\ u_k f_0 + u_0 f_k \end{pmatrix}. \end{array} \right.$$

このとき，畳み込み行列によるシルベスター行列 $\mathcal{S}_r(f_0, \ldots, f_k)$ は，次のような形になる。

$$\mathcal{S}_r(f_0, \ldots, f_k) = \begin{pmatrix} C_{n_0 - r}(f_1) & C_{n_1 - r}(f_0) & \vec{0} & \cdots & \vec{0} \\ C_{n_0 - r}(f_2) & \vec{0} & C_{n_2 - r}(f_0) & \cdots & \vec{0} \\ \vdots & & & \ddots & \vdots \\ C_{n_0 - r}(f_k) & \vec{0} & \cdots & \vec{0} & C_{n_k - r}(f_0) \end{pmatrix}.$$

3.2 計算コストの増大に関する解析

前述のアルゴリズムと方法で数式の簡単化を行うことが出来るものの，取り扱う多項式の数の増加に伴い，小さいサイズの数式の簡単化においても計算コストが大きく実用的な速度とならない。そこで，ここではアルゴリズムのボトルネックはどこかについて取り上げる。整数係数多項式の近似 GCD を求めるアルゴリズムを端的に記述し直すと次のように，何度も格子算法 (LLL) を使わなければならないことに気がつく。

1. 許容度を大きくしつつ以下を実行
2. 部分集結式の次数をあげつつ以下を実行
3. 行列の係数部分を大きくしつつ以下を実行
4. 格子算法 (LLL) で短いベクトルを計算
5. 余因子候補のそれぞれに対して以下を実行
6. 格子算法 (LLL) で短いベクトルを計算

しかし，格子算法はそもそも遅く，LLL アルゴリズムの計算量は，$\mathcal{O}(n^4 logA)$ （n はベクトルのサイズ，A はベクトルの大きさ）であり，かつ多変数かつ複数多項式の部分集結式行列は巨大（n が巨大） である。その結果，非常に簡単な数式の簡単化を行うのにも多大な時間がかかってしまう。

3.2.1 実験に使用した処理系の効率の比較

本稿では，ISSAC 2008 のポスターセッションで実験に使用した *Mathematica* 6 用の実装を利用して数式の簡単化について検討している。そこで，*Mathematica* 6 の格子算法の実装が遅いために計算時間をいたずらに長くなっていないかを， Victor Shoup の C++ライブラリである ntl-5.4.2 と比較することで確認してみる。Wolfram Research Inc. のウェブサイトによれば，*Mathematica* の格子算法を実現する組込み関数 LatticeReduce は，Lenstra-Lenstra-Lovasz による LLL アルゴリズムではなく，その Storjohann[4] による改善アルゴリズムを利用している。このアルゴリズムは LLL に比べて計算コストが小さく，$\mathcal{O}(n^3 logA)$ である。一方，ntl-5.4.2 には，整数演算での格子算法ルーチン（LLL）と，倍精度浮動小数点数を利用するルーチン（LLL_FP）とが用意されている。前者は Cohen の書籍に記載のアルゴリズムを利用しており，後者は Schnorr と Euchner による結果 [5] の亜種を利用している。比較に用いたのは，次の 3 つの多項式である。

$$
\begin{aligned}
f(x,y) = \quad & (8y^2x^2 + 3yx^2 + x^2 - 7y^2x + 7yx + x + 9y^2 + 6y - 1) \\
& (8y^2x^2 - 4yx^2 + 9x^2 + 2y^2x - 2yx - 3x - 4y^2 + 5) \\
& + y^4x^4 + y^3x^4 - yx^4 - x^4 - y^4x^3 - y^3x^3 + y^4x^2 + y^3x^2 \\
& + y^2x^2 + yx^2 - x^2 + y^4x + y^2x + x + y^4 - y^3 + y^2 - y + 1
\end{aligned}
$$

$$
\begin{aligned}
g(x,y) = \quad & (8y^2x^2 + 3yx^2 + x^2 - 7y^2x + 7yx + x + 9y^2 + 6y - 1) \\
& (8y^2x^2 + 9yx^2 - 4x^2 + 5y^2x - 9yx + 9x + 5y^2 + 9y - 2) \\
& - y^4x^4 - y^2x^4 + y^3x^3 - x^3 - y^4x^2 + y^3x^2 - y^2x^2 \\
& - yx^2 + y^4x + y^3x - y^2x + x - y^3 + y^2 - y
\end{aligned}
$$

$$
\begin{aligned}
h(x,y) = \quad & (8y^2x^2 + 3yx^2 + x^2 - 7y^2x + 7yx + x + 9y^2 + 6y - 1) \\
& (9y^2x^2 + 4yx^2 - 5x^2 + 9y^2x - 9yx - 8x + 4y^2 + 10y) \\
& + y^3x^4 - yx^4 - x^4 - y^4x^3 - y^3x^3 - y^4x^2 - y^3x^2 \\
& + yx^2 - x^2 - y^2x - y^4 - y^3 - y + 1
\end{aligned}
$$

これら3つの多項式 $(f(x,y), g(x,y), h(x,y))$ の近似GCDを計算する際には，次のような大きな行列の行ベクトルで生成される整数格子における短いベクトルを計算する必要が生じる。

- $\mathbb{Z}^{108\times 380}$ (380次元ベクトル108個で生成される整数格子)
- 絶対値最大の要素は，730185381916075706072607421875

この計算に *Mathematica* 6 の LatticeReduce は「52.5313秒」かかり，GCCでコンパイルした ntl-5.4.2 の LLL_FP は「0.400025秒」かかった。ntl-5.4.2には，検算や厳密演算のルーチンLLLとの比較を行うサンプルコードも含まれており，これらも含めると，259.236秒かかることからLLL_FPの速さが際立っていることがわかる。なお，計算時間はAMD Athlon(tm) XP 2500+，メモリ1GBのLinuxマシンで計測した。$\mathbb{Z}^{155\times 555}$ で最大5041の整数格子でも試してみたが，LatticeReduceだと「37.1823秒」かかり，LLL_FPでは「2.01213秒」で計算が完了した。これらから分かるように，*Mathematica* の組込み関数LatticeReduceは厳密演算であることを考慮すれば遅くないが，浮動小数点数を活用した高速アルゴリズムには完敗であることがわかる。

4 まとめ

摂動を整数上に制限した近似代数演算による数式の簡単化を，人間らしい簡単化と考えると，比較的望ましい結果が得られることがわかった。また，近似因数分解でなくとも，近似GCDでも簡単化が可能であることも示せた。本稿の実験結果から，多変数複数多項式の整数上の近似GCDは計算コストが大きく，かなり遅い演算であることがボトルネックであり，一連のポスター発表等で利用した *Mathematica* の格子算法は厳密演算を行うために速くないことも実験から判明したが，仮に処理系をntl-5.4.2に変えることで，10倍から100倍以上の高速化が可能であることも推測できた。しかしながら，2つの多項式でも格子のサイズを決定するシルベスター行列が，

$$Syl_r(f,g) = (C_{m-r}(f) \mid C_{n-r}(g)) \in \mathbb{Z}^{(\beta_{n+m-1,r})\times(\beta_{m-1,r}+\beta_{n-1,r})}$$

となることから，処理系を変えるなどして格子算法を高速化しても，数式の簡単化という立場からは速度の本質的な改善にはならず，整数係数多項式の近似代数演算の抜本的な高速化が今後の課題である。

参 考 文 献

[1] K. Nagasaka. Approximate Polynomial GCD over Integers. *ACM Communications in Computer Algebra*, **42(3)**. *Abstracts from the Poster Session, ISSAC 2008*. 2008, 124–126.

[2] Lenstra, A. K., Lenstra, Jr., H. W., Lovász, L. Factoring polynomials with rational coefficients. *Math. Ann.*, **261 (4)**. 1982, 515–534.

[3] Rupprecht, D. An algorithm for computing certified approximate GCD of n univariate polynomials. *J. Pure Appl. Algebra*, **139 (1-3)**. 1999, 255–284.

[4] Storjohann, A. Faster Algorithms for Integer Lattice Basis Reduction. *Technical Report* **249**. Zurich, Switzerland: Department Informatik, ETH. 1996.

[5] Schnorr, C. P., Euchner, M. Lattice basis reduction: improved algorithms and solving subset sum problems. In: Proceedings of Fundamentals of Computation Theory, FCT'91, Ed. L. Budach, *Springer LNCS* **529**. 1991, 68–85.